

For Reference

NOT TO BE TAKEN FROM THIS ROOM

Thesis
2F-198

Ex LIBRIS
UNIVERSITATIS
ALBERTAENSIS





Digitized by the Internet Archive
in 2021 with funding from
University of Alberta Libraries

<https://archive.org/details/Rushton1972>

THE UNIVERSITY OF ALBERTA

A CRITIQUE OF PROGRAMMING TECHNIQUES
FOR PLAYING CHESS

BY



PAUL RUSHTON

A THESIS

SUBMITTED TO THE FACULTY OF GRADUATE STUDIES AND RESEARCH
IN PARTIAL FULFILLMENT OF THE REQUIREMENTS FOR THE DEGREE OF
MASTER OF SCIENCE

DEPARTMENT OF COMPUTING SCIENCE

EDMONTON, ALBERTA

FALL, 1972

THE UNIVERSITY OF ALBERTA

FACULTY OF GRADUATE STUDIES AND RESEARCH

The undersigned certify that they have read, and recommend to the Faculty of Graduate Studies and Research for acceptance, a thesis entitled A CRITIQUE OF PROGRAMMING TECHNIQUES FOR PLAYING CHESS submitted by Paul Rushton in partial fulfillment of the requirements for the degree of Master of Science.

Abstract

A discussion of approaches used in the design of a machine to play chess is presented. The brief history of chess playing programs is included in highlight form. No attempt is made to describe most written programs. Rather, the important techniques used in each program have been extracted and discussed under appropriate subtopics. It is felt that these methods are insufficient for producing very strong mechanical players and reasons for these convictions are detailed. An alternative approach along goal directed lines is suggested. Although this method has been attempted previously, and failed, new ideas are presented to overcome the former areas of difficulty. The specifications for a program operating along these novel lines are outlined. Finally, a brief overview of the field is given and suggestions made for future research.

Acknowledgements

I wish to thank both Professor T.A. Marsland, my supervisor, and Dr. L.K. Schubert for their advice and guidance throughout the preparation of this research.

The financial assistance received from the National Research Council of Canada is also gratefully acknowledged.

Table of Contents

	<u>Page</u>
Chapter 1 INTRODUCTION	1
Chapter 2 HISTORY	4
2.1 The Early Beginnings	4
2.2 Shannon's Work	4
2.3 From 1950 to 1958	6
2.4 Newell, Shaw, Simon	6
2.5 From 1958 to 1972	7
2.5.1 Scoring Function	7
2.5.2 Tree Building	9
2.5.3 Termination Criteria	11
2.5.4 Backing-Up	12
2.5.5 Learning	17
2.6 Current Programs	18
2.6.1 Greenblatt	18
2.6.2 Gillogly	19
2.7 Discussion	20
Chapter 3 DEFICIENCIES OF CURRENT PRACTICES	21
3.1 Problem Areas	21
3.2 Evaluation Function	21
3.3 Tree Building	26
3.4 Backing-Up	29

Table of Contents (Cont.)

	<u>Page</u>
3.5 Final Choice	29
3.6 Experience Problem	30
3.7 Conclusions	30
 Chapter 4 GOAL-DIRECTED METHODS	 32
4.1 Goal Approach	32
4.2 Previous Approach	33
4.3 General Description of the System	35
4.3.1 Feature Recognition Programs	36
4.3.2 Goal Statements	38
4.3.3 Planning Programs	38
4.4 System Operation	41
4.5 Conclusions	43
 Chapter 5 PAST, PRESENT AND FUTURE	 44
5.1 Past	44
5.2 Present	44
5.3 Future	45
5.4 Conclusions	46
 References	 47
Appendix A The Alpha-Beta Algorithm	51
Appendix B Construction of Various Scoring Functions ...	52

List of Figures

<u>Figure</u>		<u>Page</u>
1	Multistage Signature Tables	10
2	Minimax Backing-Up Procedure	13
3	Examples of an Alpha and Beta Cutoff	14
4	Minimax	16
5	2 & 2	16
6	Utility of Several Scoring Functions	24
7	Sample Game Tree Using Short Term Goals	28

Chapter 1

INTRODUCTION

The purpose of this thesis is to examine methods for playing chess and discuss their likelihood for success. Although such work may not be of much practical use by itself, it is felt that the methods used to produce a good chess playing program are applicable to other useful areas. A list of these applications is found in Shannon's 1950 paper [16]. The goal of artificial intelligence is to create programs which exhibit behaviour that would be called intelligent if performed by man. Toward this end, a strong chess playing program is worthwhile.

There are several advantages in using the game of chess as an example. Its moves and the goal of checkmate are well-defined. The game is not simple; in fact it has supported over one hundred years of extensive analysis and has still not been exhausted.

Another advantage of chess is that a satisfactory method exists for determining the strength of a player. A player is rated depending on his successes and failures against other rated players. After a set number of games a player is assigned a provisional score. Thereafter, a player's rating changes according to the outcome of the game and the difference between his own rating and that of his opponent. When a person achieves a rating of 2200 he is

classified as a master. Out of 1450 rated Canadian players only 1.5 per cent are of master strength. Current programs have ratings in the vicinity of 1500, a figure exceeded by half of all rated Canadians. Thus programs have a long way to go before reaching the master class.

There is a constraint on chess which is peculiar to that game. Each player must make a preset number of moves within a specified time limit, such as 40 moves in 2 1/2 hours. It imposes a unique requirement on a program to play the game.

Along with the time limit, the fact that chess playing programs use many different computers, which in turn operate under varying conditions, makes it difficult to compare the separate algorithms. Thus a slightly inferior method using a devoted machine might perform better than a superior one running on a time-shared system. For purposes of comparison and evaluation of algorithms it seems reasonable to replace the strict time constraint of tournament chess with a more relaxed one. A program should be allowed to make a move in an hour or so. Such an amount of time would allow the program to select the move it considers best. On this basis an accurate rating could be determined for the program which would be comparable to ratings of people and other programs. Once a master calibre program was developed in accordance with the above freer limit it could be coded more efficiently and/or run on a more efficient computer to conform with tournament rules.

Game playing programs usually build some kind of tree structure when selecting a move. A clarification of terms pertaining to this entity is in order. A move in chess generally means one white move followed by one black move. This is not too useful a term when used in conjunction with game playing programs and so the terms "ply" and "half-move" have been introduced to refer to either a white move or a black move. Thus two plies or two half-moves equal a move in the general sense. Game trees may be of several kinds. A "complete game tree" is the one containing all legal variations whose end nodes are terminal as defined by the rules of the game. An "explicit game tree" is that part of the complete game tree a program actually explores.

Chapter 2

HISTORY

2.1 The Early Beginnings

Interest in designing a machine to play chess originated a long time ago. During the late 18th and early 19th centuries von Kempelen (see [16]) exhibited his Maelzel Chess Automaton, supposedly a machine capable of playing master strength chess. It proved to be a well concealed midget. Many years later, in 1914, Torres y Quevedo (see [16]) built a device to play the rook and king versus king end game. Although this problem was quite simple, Quevedo was ahead of his time in conceiving it.

2.2 Shannon's Work

It would not be unfair to say that the history of chess playing programs began with the paper written by Shannon in 1950 [16]. In his article one can find many basic concepts underlying the operation of current game playing programs. What follows is a summary of these important points.

First of all it has been shown many times that the complete game tree for chess can become extremely large and so it is desirable, in fact imperative, that the size of the tree searched or constructed be kept small. In each of the first twenty complete moves of a game each player has a choice of roughly 35 possible moves. Thus there are

approximately 35^{40} variations in a game tree for these first moves. Second, one must have some means for evaluating moves. For this purpose Shannon suggested the use of a polynomial scoring function whose terms reflected the relative merit of various chess heuristics. Material advantage for instance, usually enables one to win the game so that this term would appear with a positive coefficient in the evaluation function, whereas weaknesses such as isolated pawns would have terms associated with negative weights. Third, assuming that this kind of function would be used, Shannon went on to indicate that it should only be employed in "quiescent" positions. A quiescent position may be defined as one in which no move exists which will radically alter the value of the scoring function. For example, if material advantage was being calculated then there would be no point in performing this computation during capture sequences. The position would not be quiescent during this series of events. Fourth, the idea of using the minimax backing-up procedure was mentioned. This is the name given to the following method. One assumes the program will try to maximize its gain while the opponent attempts to minimize it. Action is then taken in accordance with this principle, when searching the tree structure for the move to be made. Fifth, Shannon also proposed that a program might be made to learn by modifying the coefficients of its scoring function. A sixth suggestion was separation of phases of the game and writing programs to cope with each

of these individually. Seventh and finally, he believed that a much superior chess program could be written along lines similar to the way people play. These ideas have proved fundamental not only to chess but more generally to the game playing area of artificial intelligence.

2.3 From 1950 to 1958

At Los Alamos (see [11]), a program was written to play chess on a 6X6 board. The bishops were not used in this game and all special moves were eliminated. Evaluation was based on the sum of material and mobility and was applied to the terminal nodes of an exhaustive four ply tree. The program was capable of beating only a beginner.

Later, Bernstein (see [11]) wrote a program for the full 8X8 board which also built a tree of four plies. However, not all possible moves were considered. A number of chess heuristics were used to retain only a few legal moves for further analysis. The factors of material, king safety, mobility and area control were weighted and summed for each side at the terminal positions. The ratio of these sums constituted the evaluation function. This program, too, was weak and could defeat only a novice.

2.4 Newell, Shaw, Simon

The next important contribution to ideas about game playing came in 1958 from Newell, et al. [11]. The intervening years saw the development of several chess

programs built around Shannon's principles. These programs did not introduce new concepts.

Newell, Shaw and Simon introduced a goal-directed approach. A set of goals was selected by a preliminary analysis routine which in turn controlled the remainder of the move selection process. Goal methods will be considered more fully in Chapter 4.

2.5 From 1958 to 1972

No fundamentally new ideas have been proposed since 1958. Several programs of noteworthy ability were constructed, namely CHESS 3.5 [19], COKO III [9], TECH [2] and The Greenblatt Chess Program [4]. The latter two will be described in Section 2.6. Most of the programs of this period have ratings in the range 1000-1500. They play the opening in a passable fashion and perform about the same or slightly better in the middle game. All of them are characterized by weak end game play.

Rather than describe in detail the programs written during the last fourteen years, the techniques used will be discussed. Since many algorithms employ similar methods, a precis of each program would be needlessly redundant.

2.5.1 Scoring Function

As mentioned previously, Shannon first introduced the concept of a scoring function and it has since been universally employed in game playing programs. Such a

function consists of a number of different terms which try to measure the degree to which certain features are present. The features are based on principles of play and, depending on their nature, their presence generally indicates an advantage or disadvantage which is reflected in the value of the evaluation function.

The above means of measurement may not take into account interaction between two features very well. For instance the combined appearance of two or more features is not necessarily the sum of the individual feature values. There is also the problem of a combinatorial explosion of terms if all combinations are considered. For example, with 10 original features there are $2^{10}=1024$ possible interactions. To take these problems into account Samuel [15] uses what he calls "signature tables". Although this technique is not currently employed in chess playing programs it appears worthy of inclusion because of its general applicability (see also B.3).

Signature tables enable one to consider relationships among features as indicated in the following example. First of all, one must select a fixed number of features, five in this example. Assume these are assigned values (always with a small finite range to ensure the signature table does not need to be too large) of 2, 1, 2, 0, 0, respectively. The value 21200 base 3 is used as an index to a table of values (the signature table) which in turn gives the score for this combination of features. In general, multiple sets of

features are employed. Values for each of these combinations may be used as yet another index to another signature table which yields the final score. (See Fig. 1)

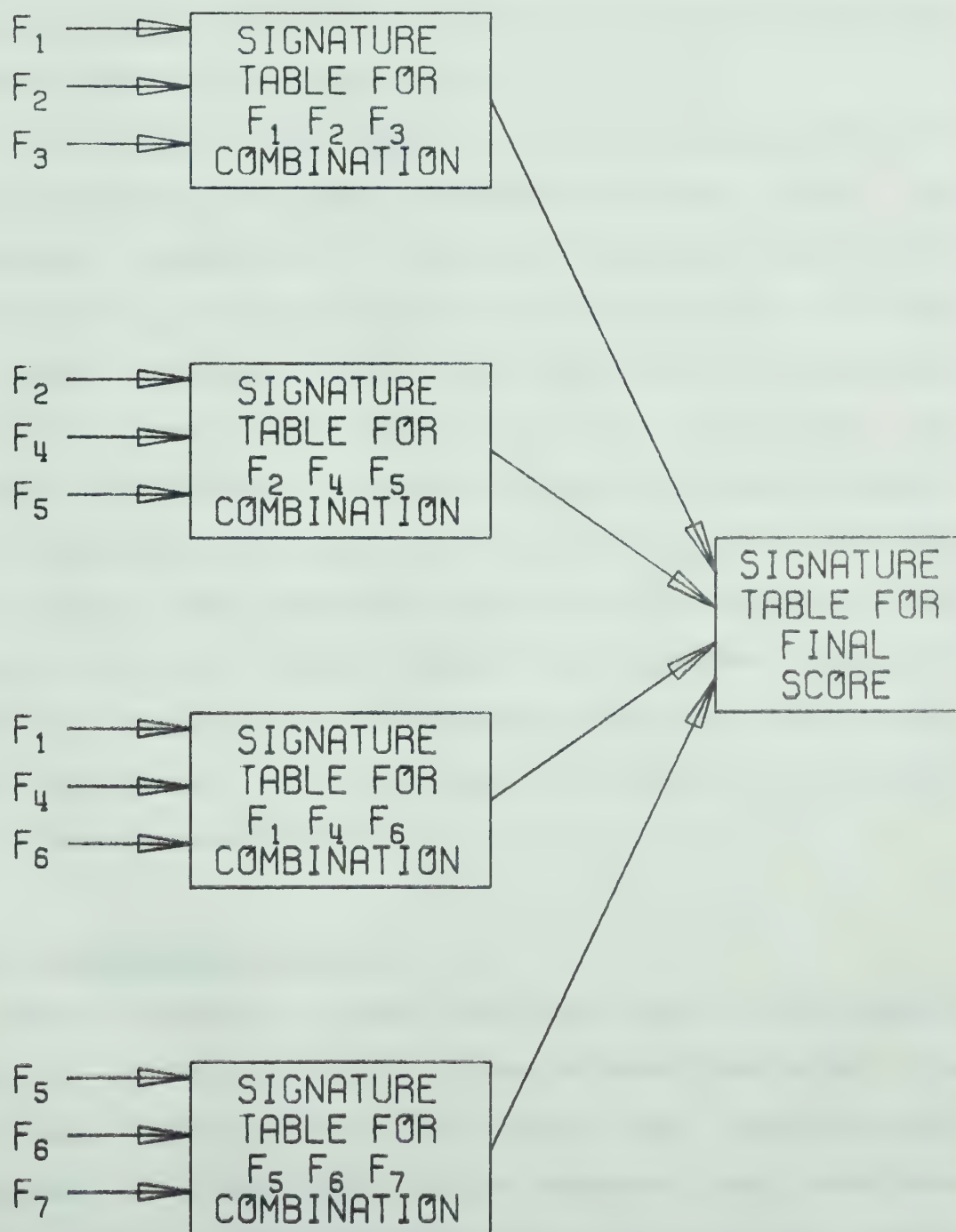
This method seems quite worthwhile and is probably useful in many game playing settings. There is no predetermined manner for selecting the sets of features, however, and no method is currently available for having the program determine them. The decision is entirely up to the programmer.

2.5.2 Tree Building

There are an assorted number of ways to search a game tree [17, pages 13-20]; the most straightforward is to build a tree to a preset depth and consider all legal moves from each position. Since trees have a tendency to grow exponentially with depth, this is generally not the most advisable method. Other techniques have been devised and a summary of these will be given.

Shannon suggested decreasing the number of moves examined from each position with increasing depth in the tree. Thus, one may analyse ten legal moves from the initial position, eight from its successors, etc. Slagle called this "tapered n-best forward pruning". The next two methods are closely allied with the termination criteria (see section 2.5.3). "Selective search" sprouts a tree from the seemingly best continuation, as measured by

FIG. 1 MULTISTAGE SIGNATURE TABLES



some method, until the value for a particular variation exceeds some predefined value. The goal directed techniques of Newell et al. [11] explored variations until quiescence was reached and the value for that move sequence did not exceed a threshold value. Many slight variants to the above tree searching methods exist but the preceding discussion outlines the ones in general use.

Botvinnik [1] has suggested a new means by which it would be possible to limit the size of the game tree. The underlying assumption is that both sides are striving for material gain. One must consider all attacks and defences of pieces occurring within a fixed number of plies and within this domain (Botvinnik calls this an horizon) the problem of material advantage should be solved exactly. Thus if the horizon is set at 4 ply then one must determine all attacks and defences that occur within this limit and select the move which results in the most favourable position. To date, the practicability of this idea has not been tested, or at least no information is available concerning its applicability.

2.5.3 Termination Criteria

When building a tree one must always decide when to stop. It is the termination criteria which determine this. The usual ones employed are game over, maximum depth, quiescent position, time limit exceeded and acceptance value. These are self-explanatory with the possible

exception of the last one, in which one keeps building the tree until a move receives a backed-up score exceeding the acceptance value. Not all of these criteria must be employed of course; but it is obviously necessary to have some stopping condition.

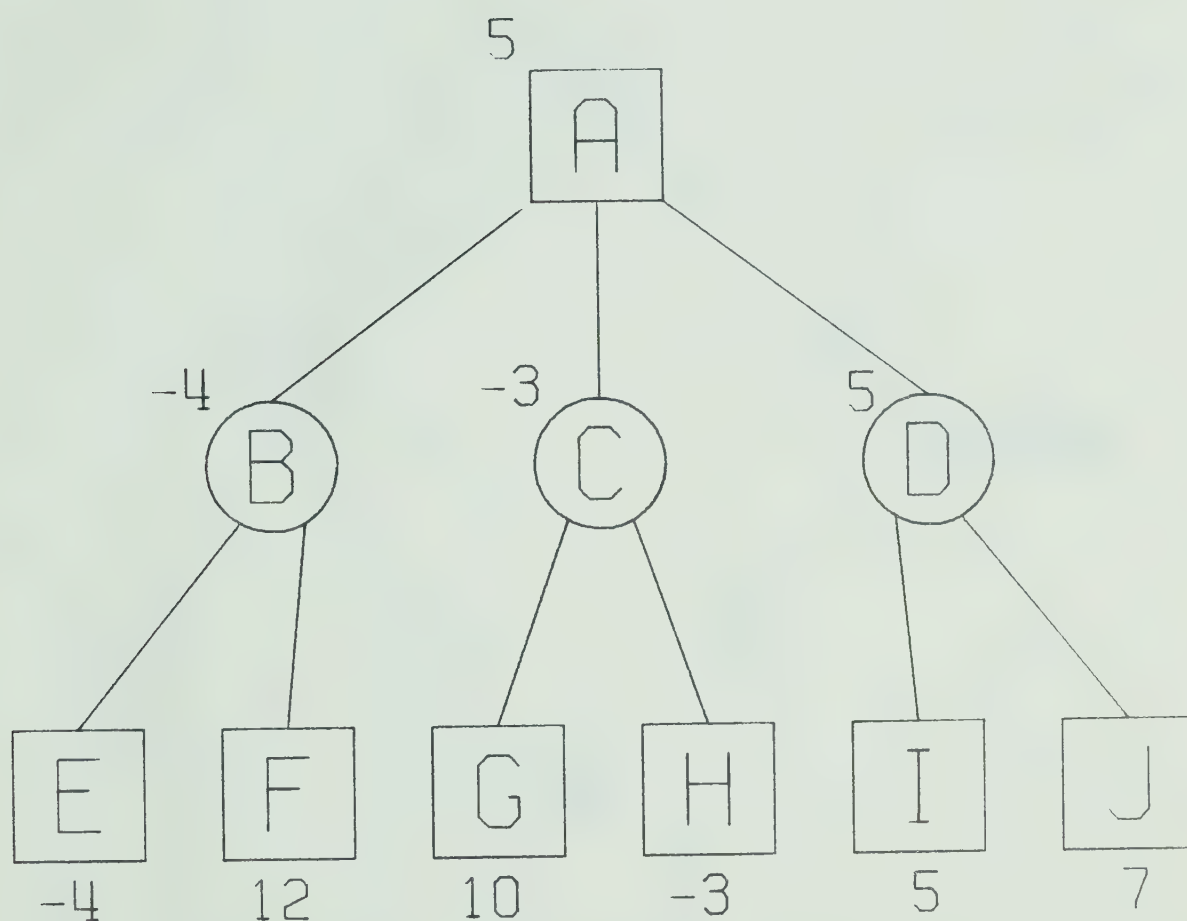
2.5.4 Backing-Up

Once the tree has been completed one must select a move to be made from the initial position. The method already mentioned assumes the program will try to maximize its gains while the opponent attempts to minimize them. In the following discussion MAX denotes the maximizing player and MIN the minimizing one.

In Fig. 2 at the nodes B, C, D, MIN will select the moves to nodes E, H, I respectively. At the initial position, A, MAX will choose the best alternative, which is the move to D. This procedure, the minimax one, is widely used in game playing programs but can be modified to work more effectively.

In Fig. 3 the values at nodes G and H are 1 and -10 respectively, so that the value at D is 1. When this score is assigned we know that at B, MIN can force a continuation to have the value 1 at most. At E, MAX has a choice which scores at least 2. This immediately implies that MIN will not choose the path from B to E, so that further exploration from E (to J for example) is unnecessary. A beta cut-off is said to occur at E. Similarly, an alpha cut-off occurs at C.

FIG. 2 MINIMAX BACKING-UP PROCEDURE



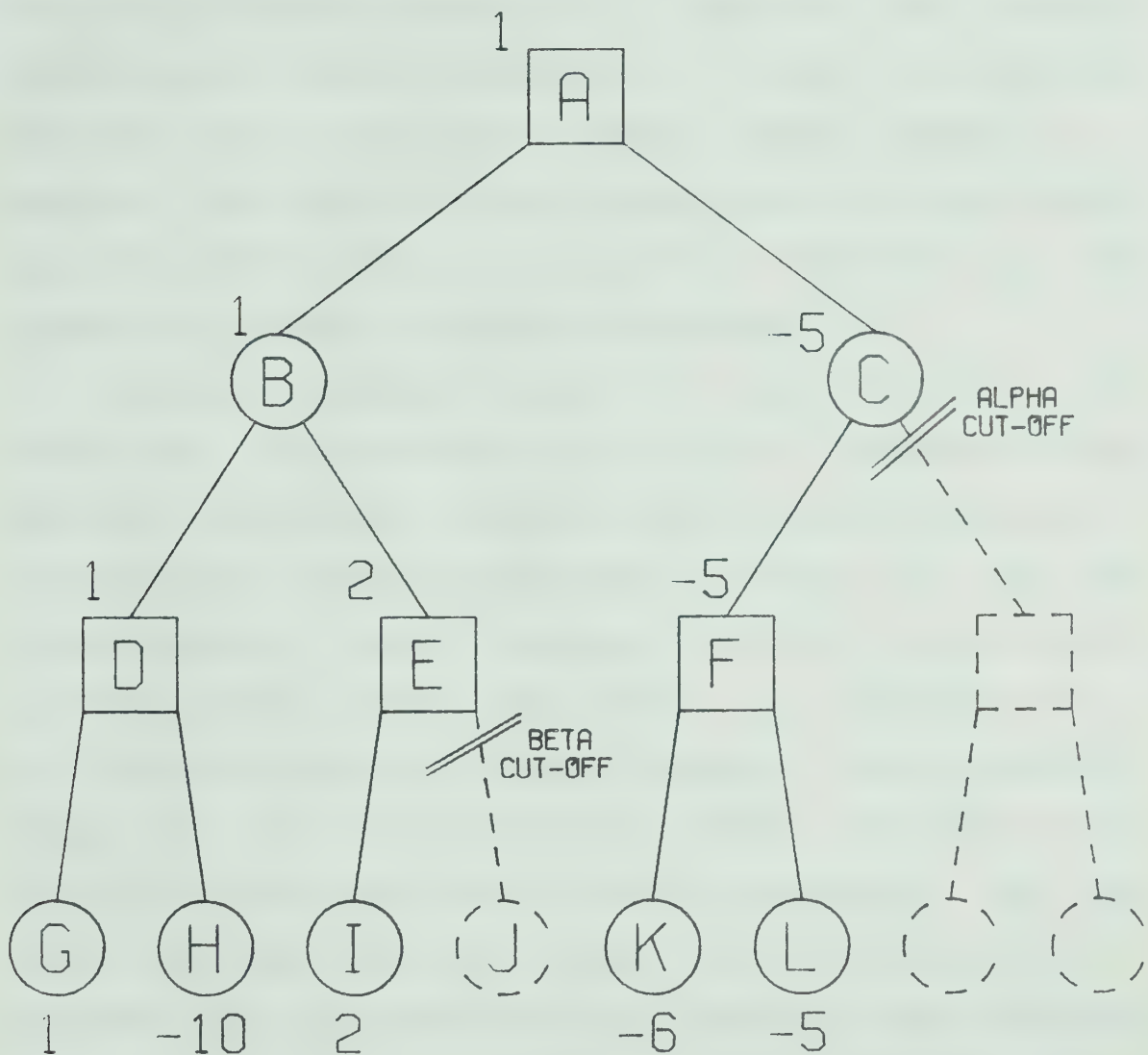
DENOTES MAX NODE



DENOTES MIN NODE

NUMBERS ARE VALUES FOR THE POSITION

FIG. 3 EXAMPLES OF AN ALPHA AND BETA CUTOFF



DENOTES MAX NODE



DENOTES MIN NODE

NUMBERS ARE VALUES FOR THE POSITION
 DASHED ENTITIES ARE UNEXPLORED BRANCHES

Here, MAX can force a variation to have a value of at least 1 from node A, while at C MIN can select a move sequence resulting in a score of -5. Thus, MAX will not move to C and so the remaining moves from C need not be explored. This revised minimax principle is called the alpha-beta procedure and its employment almost always results in a smaller tree search than the minimax one, yet allows one to make the same decision. A more precise definition of the alpha-beta procedure is given in Appendix A.

Another backing-up method is given by Slagle and Dixon [18]. This technique is the M & N procedure which operates as follows. Assuming that the scoring function is unreliable this method assigns some function of the M(N) highest(lowest) valued successor nodes as a value for the original MAX(MIN) node. Thus minimax is equivalent to a 1 & 1 procedure with an identity function. One can argue in favour of the M & N procedure as follows. At the terminal nodes of a tree, the score is only an estimate of the true value for the position. Thus, several successors of a MAX(MIN) node with nearly the same high(low) value indicates that node is more desirable than a MAX(MIN) node with only a single high(low) valued successor.

One can see the difference between the minimax and 2 & 2 procedure by comparing Fig. 4 and Fig. 5. The 2 & 2 procedure backs up the average of the best(worst) two successor values for a MAX(MIN) node. Note that in general

FIG. 4 MINIMAX

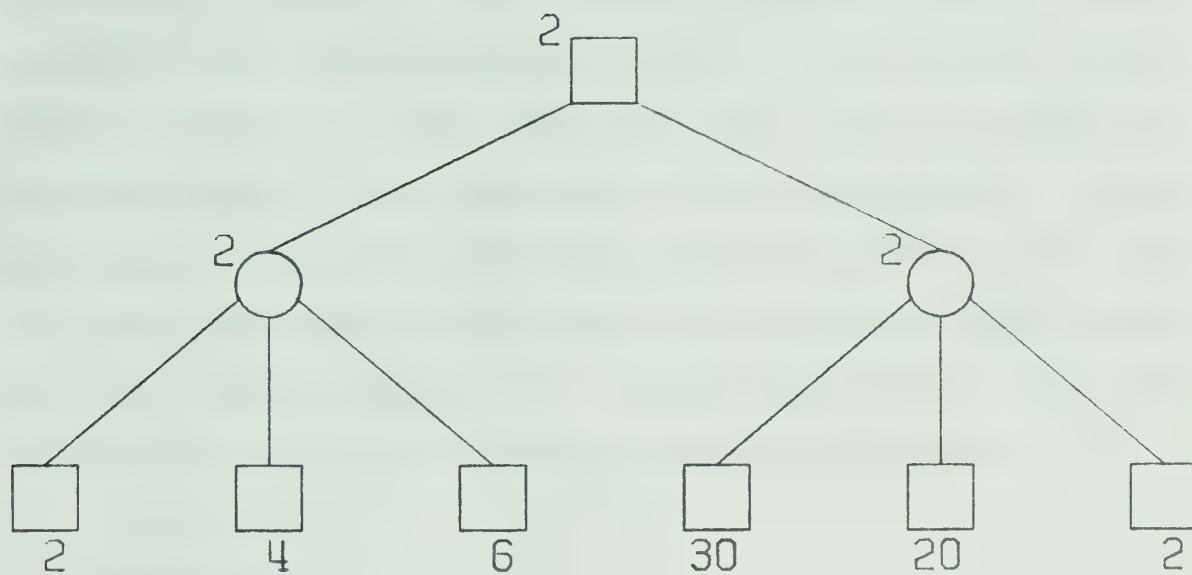
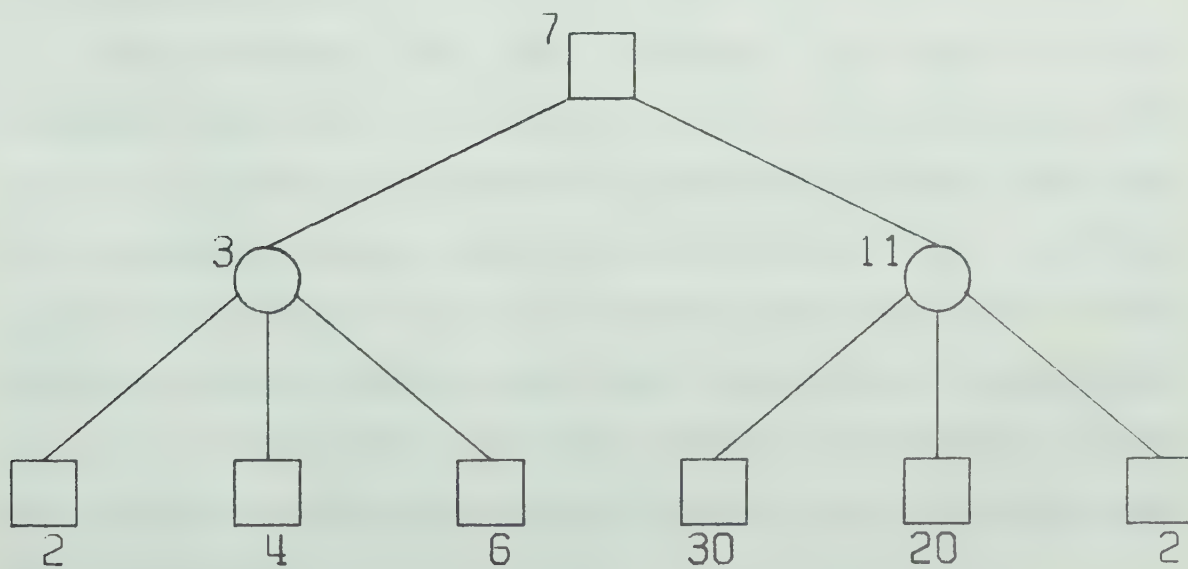


FIG. 5 2 & 2



any function of the successors could be chosen. Since the terminal scores in the right hand branch have a much higher mean than those in the left one the right branch should be preferred. However, the minimax method has no way to indicate that the right hand branch of the tree may be better than the left, whereas the 2 & 2 procedure most certainly does. This procedure has not been used in chess; however, it has given extremely favourable results when used to play kalah and there seems to be no reason why it would not give similar results in chess, especially since the scoring functions have a large degree of uncertainty.

2.5.5 Learning

The work that has been done in the area of learning with respect to game playing has been very limited. Only two approaches to the problem have been used to date. One is rote learning, the other is coefficient modification.

Rote learning has been employed by Samuel [14] in checkers and Slate et al. [19] in chess. To accomplish this form of learning one keeps the entire position, along with its associated value, and then searches a list of these positions during the move selection process. Only in the opening phase of the game has rote learning been successful. Later on in the middle game the number of positions which can occur is simply too large to allow use of this facility in either game.

The other approach to learning has been the adjustment of coefficients of terms in the scoring function. Several methods (see [17, Chapter 11]) are available for doing this but no dramatic increase in program performance has been observed. It appears to be only a "fine tuning" technique for programs using a scoring function.

2.6 Current Programs

The present state of the art is exemplified by two programs. Both of these will be described.

2.6.1 Greenblatt

Mac Hack, the program written by Greenblatt, et al. [4] is one of the best playing programs. Over the years it has improved in ability but this has not been due to a learning capacity. An evolutionary process of additional programming of chess heuristics has been responsible.

For any position, all legal moves are listed and assigned a plausibility score. Some of the heuristics that influence this value are as follows. Moves to the centre or in the vicinity of the opponent's king are given preference. Also, moves which block the enemy's pieces or put them en prise are considered valuable. A subset of the plausible moves is then considered for deeper analysis. Each move is the root node for a new tree. The termination criteria are maximum depth and game over. When either condition is satisfied a score for the position is calculated. This

scoring polynomial is different from the plausibility function and consists of terms reflecting material balance, pawn structure, piece ratio, king safety and centre control. The resultant value is backed-up to the initial position by the alpha-beta algorithm and the final choice is the move which receives the highest score.

The program plays quite well in the opening and middle games. Its rating is near 1500 which is close to the mean of all rated Canadian players. As noted before, the end game play is weak and it is doubtful if Mac Hack can win with a rook and king versus king. The Greenblatt program is a good example of Shannon's ideas put into practice.

2.6.2 Gillogly

Gillogly's program [2], TECH, is representative of the set of programs which build a large tree but use a simple scoring function. No heuristic pruning other than alpha-beta is used on its game tree. The scoring function consists solely of material difference. All legal moves are considered from each position and the tree is built to a fixed depth with capture sequences explored beyond this limit. Again, the alpha-beta procedure is used to back-up the score. Chess heuristics are employed to order the successors of the initial position, however this ordering takes place only at the root node. TECH's play is similar to that of Mac Hack. Gillogly's method indicates how well a program, based on computer speed alone, can perform.

2.7 Discussion

A brief history of chess playing by machines was given and the current state of the art summarized. Two approaches to the problem were evident; one was goal-oriented the other was built around the concept of an evaluation function. Within this latter category yet another division is worth mentioning. There is a trade-off between complexity of a scoring function and selectivity, as illustrated by the comparison between Mac Hack and TECH. This compromise has not been investigated fully but is worthy of attention since TECH placed second in the second U.S. Computer Chess Championship ahead of other programs using a sophisticated scoring function and higher selectivity. There is a general tendency for the better playing programs to become more complex with each increase in ability. This may prove to be the limiting factor in their strength if all improvements are due to additional human programming rather than machine learning.

Chapter 3

DEFICIENCIES OF CURRENT PRACTICES

3.1 Problem Areas

Three main areas can be identified in which difficulties may arise - the evaluation function , the tree building mechanism, and the backing-up procedure. The scoring function in conjunction with the tree building procedure seem to be the source of most problems. An indication is given by the fact that this method for playing games has been in use in chess for nearly twenty years and still has not produced a master strength program. For the most part, little fault can be found with backing-up methods insofar as the minimax assumption is valid.

3.2 Evaluation Function

The use of a conventional scoring function for playing chess is unlikely to produce a master calibre program. The present theory of evaluation functions is not sufficient to prove the truth of the preceding remark, but much empirical evidence indicates its validity. It is hoped that the following discussion will bring the use of an evaluation function under serious criticism.

Implicit in the use of a scoring polynomial are the following assumptions. Every position can be decomposed into separate features. In turn, each feature can be

assigned a value reflecting its degree of presence. Finally, the resultant scores can be recombined by some function to represent the merit of the original position.

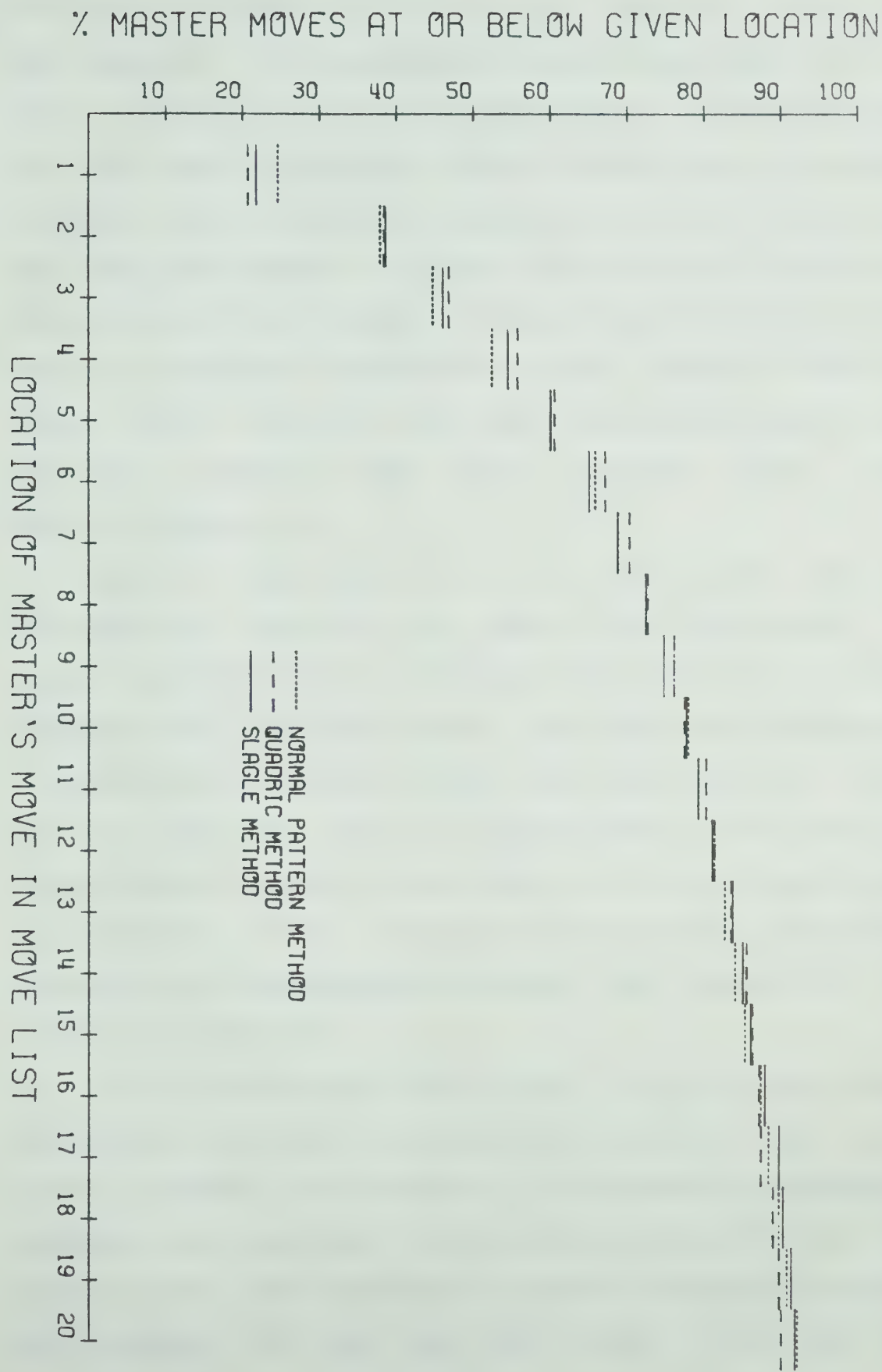
Presupposing the above assumptions correct, let us consider the terms which should comprise an acceptable evaluation function for the middle game. Point Count Chess [8] and Chess Made Simple [6] furnish approximately forty-five principles whereas in My System [13] about ninety are discernible. There is a subdivision of principles in the latter book which are taken as one in the other references. Since these concepts have been expounded by master class players it is reasonable to assume the scoring function should base its score on them if it is to have any chance of performing at a high level.

Assume these principles are the only ones of importance for this stage of the game. When writing a program to assess accurately some of the features, one may be confronted with programming problems. For instance, rook on the seventh rank and passed pawn are easy features to compute; however, features such as control of the centre and control of a useful open file are much more difficult since the words "control" and "useful" are not well-defined in the chess literature. As such, poorly defined terms are open to the programmer's interpretation and provide him with undesirable leeway. Hence, even after a program has been written, those sections of code pertaining to ill-defined features are open to question and continual modification.

A scoring function must differentiate between good and poor moves. In the middle game, de Groot [5] found that there are approximately 35 legal moves per position and that a master class player explores variations to a depth of 7 plies. It is impossible to search all continuations (some 35⁷ of them) so some moves must be discarded. An attempt will now be made to determine the maximum number of moves per node (the branching factor) that a master class program can afford to examine. The TECH program explores a tree of at most $5 \cdot 10^5$ positions for a move in the middle game. No other program searches so large a structure. A good estimate for the branching factor for a tree of 7 plies and $5 \cdot 10^5$ nodes is 6.5. Thus, the evaluation function must assign some of the top 7 scores to the moves that masters select for analysis. One should note also that while this is a necessary condition it is not sufficient. The best move is still to be selected.

More experimental evidence against the evaluation function was obtained as follows. From The Book of the New York International Chess Tournament 1924 [7], 378 middle game chess positions and the move made in each by a master were obtained. In each position the legal move list was constructed and a value computed for each move by a scoring function without look ahead. The moves were sorted, the master's move located in this list and its position recorded. The results obtained from three evaluation functions (described in Appendix B) are shown in Fig. 6.

FIG. 6 UTILITY OF SEVERAL SCORING FUNCTIONS



At position 7 some 30 per cent of the master's moves were not selected. It is doubtful if any of these functions will form the basis of a master class program. It must be noted too that these polynomials contain many terms considered important when evaluating a position. Another indication of the weak performance of these three functions is given by their poor ability to give the master's move the best score. Finally, one should remember that these results depend on there being only one or two good moves per position on the average. Evidence for this argument comes from Good [3, Appendix D].

Several extensions to the above test would be worthwhile but time did not permit their execution. It would be desirable to determine the utility of scoring functions that used look ahead. Thus, all legal moves would be generated for shallow trees, and the backed-up scores used to sort the list of legal moves at the root node. Graphs similar to Fig. 6 would help to show the benefits of tree search. These tests could then be used with scoring functions of other programs to check the validity of the results obtained here.

It is conceivable that some or all of the initial assumptions concerning the evaluation function may be at fault. Masters tell us that it is possible to decompose a position into features. At that point however one does not assign values. The presence or absence of features controls the methods of play that are tried. Only then are

continuations explored, positions evaluated and a move selected. Although this argument does not condemn the use of an evaluation function, it should lead one to reconsider its merits in the light of its apparent ability.

3.3 Tree Building

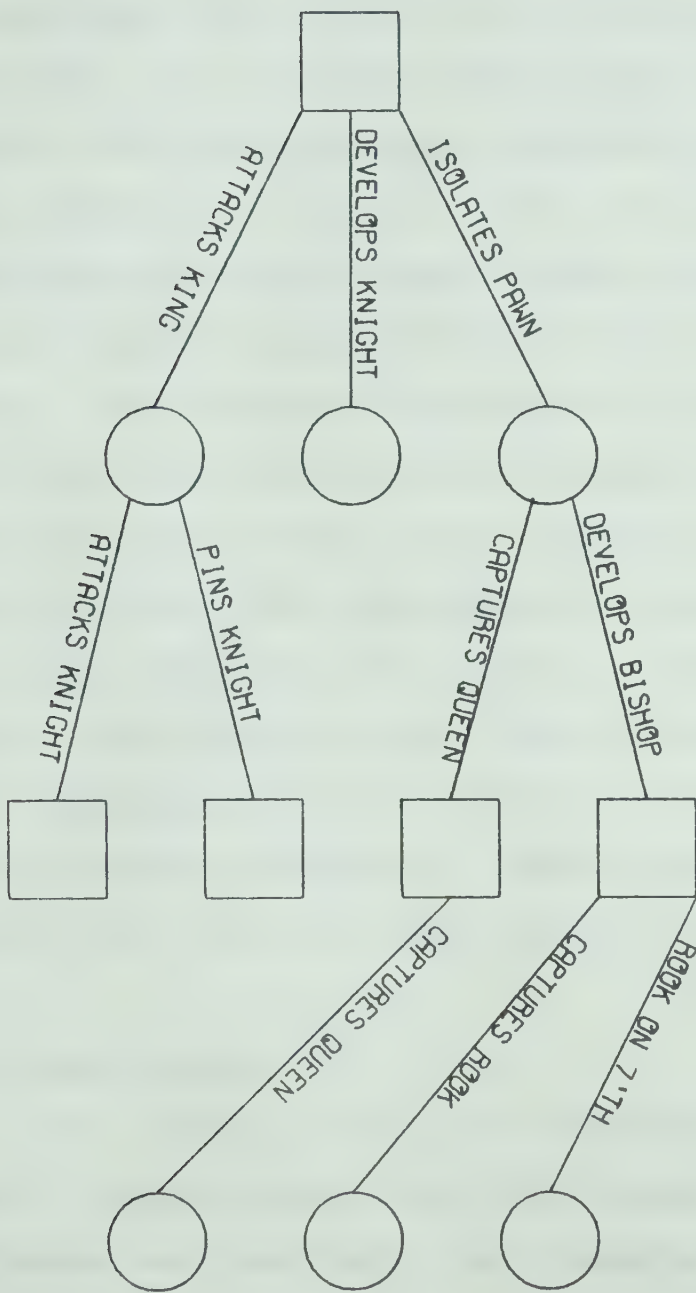
Several faults with the tree building mechanism are detectable. Since a game tree for a master class player is required to be 7 plies on the average it is mandatory that all legal moves not be explored in every position. In order to examine only a few branches from each node one must assume the evaluation function gives extremely reliable scores and search only those moves receiving the locally best score. However, some unexplored moves might result in higher backed-up scores. Thus one problem is analogous to searching a non-convex surface for a global maximum with a method that can get trapped on a local maximum.

Another major problem is the lack of continuity of planning to aid in tree construction and move selection. Planning requires an exchange of information among levels in the tree structure which is almost entirely absent in current programs. The evaluation function is used to summarize all that is known at each node in the tree structure, thus making such an exchange difficult if not impossible. Hence one does not know why certain moves were made in preceding positions which in turn eliminates any possibility of continuing the development of a previously

conceived plan. Of course, a strong program will not automatically result if the reasons for playing moves in the tree are known. It is merely a step in the right direction. For example, in Fig. 7 one knows why each move was played but no consistent plan is recognizable. Here too, little if any information is exchanged between the levels of the tree. It is necessary to utilize knowledge from other nodes in the tree structure when determining which moves to analyse. Regardless of how a planning ability can be incorporated, its absence is a most serious weakness in chess playing programs.

Botvinnik's horizon method seems appealing but it should not succeed in producing a master class program. Material gain is the ultimate goal of this method. Hence, practical computation of a move would probably require too much time. One should note that the opponent's men are considered frozen while attacks and attack paths are determined. A horizon of "n" half-moves means one has to calculate all paths of length "n" whose final half-move results in a capture. For most positions this computation is too complex unless $n \leq 3$. The method also fails under the following circumstances. Certain threats may be overlooked since the horizon method requires all variations to end with captures. It is noteworthy that Botvinnik, a former world chess champion, believes one can and should temporarily ignore the opponent's options.

FIG. 7 SAMPLE GAME TREE USING SHORT TERM GOALS



3.4 Backing-Up

There are no inherent weaknesses in either the minimax or M & N backing-up procedures, although they may credit the opponent with too much intelligence. For example, if a program sees that it will be checkmated in five moves unless it gives up a bishop (which will probably cost the game anyway) then the backing-up procedure will opt to surrender the bishop. It may in fact be better to chance being checkmated but retain equal material so that if the opponent misses the win there would be a better chance for a win or draw. The minimax technique also relies on the scores of the evaluation function being accurate. Since this is not the case the backed-up scores cannot be trusted completely. The M & N procedure compensates for this and as such should be more successful than minimax. Unfortunately this method has not been applied to the game of chess. Problems with the backing-up method could be resolved by suitable changes in the scoring polynomial. Hence, poor program performance is not due to problems with backing-up methods.

3.5 Final Choice

The selection of a move is usually done in one of two ways. Either the move with the highest backed-up score is selected or the first move receiving a score exceeding a preset threshold is chosen. Inaccuracies of the scoring function cause problems with both methods, while the second one additionally introduces the trouble of choosing an

appropriate acceptance value.

3.6 Experience Problem

To date, no program varies its behaviour according to past experience. Given an initial configuration of pieces it will make the same move every time regardless of the outcome of the game. This is not an easy problem to rectify since a program must adjust itself in some way in order to learn. Learning would be facilitated by some kind of record of past games although it may not necessarily be essential. No work of any consequence has been done with respect to learning in game playing. It will become a necessity as stronger programs are developed; otherwise, if the program lost one game it could then be beaten in the same way over and over again.

3.7 Conclusions

Central to most of the problems has been the reliability of the evaluation function. If this can be improved then a generally better performance of the entire program would be observed. The necessary degree of trustworthiness does not seem practically possible however.

The other main area of difficulty is the inability of chess algorithms to carry out a consistent plan. A means of achieving this may be to alter the weights of the evaluation function as the tree is being built. For example, if a proposed move increased the contribution of a particular

term or set of terms, then at the next level of the tree the set of coefficients associated with the preceding set of terms would be increased. A similar adjustment would be made for the terms it influenced negatively. Once a move had been actually played the scoring function would be modified as before and so would constantly change throughout the game to reflect the plans being attempted.

A program must also be able to learn from its mistakes. In order for it to do this, it has to summarize information about past games in a useful way so that it can access this data easily at some future time. It may also be possible for a program to learn by self modification based on an analysis of its performance against better players. This area is completely open for research and appears to be an integral part of any game playing system.

It appears from experience with chess programs over the past two decades that no dramatic increase in performance can be expected using present techniques. Thus it would seem worthwhile to devote attention to other methods which may overcome the preceding problem areas.

Chapter 4

GOAL-DIRECTED METHODS

4.1 Goal Approach

Since current techniques seem limited, another approach to the problem is desired. A method closer to the way people select a move in chess should be fruitful. The initial position is examined and a store of information obtained. Some goals are set and a means for achieving them planned. As this is taking place more insight into the position is gained which in turn refines the goals and the means to achieve them.

The goal-oriented techniques have several arguments in their favour. By mimicking the way humans solve problems one is assured that the methods do give excellent results if they can be copied correctly. This is not known for the evaluation function approach. A second indication of the superiority of goal methods is their capability for continuity of planning. Once a goal or several goals have been set, it should be possible to construct a much restricted game tree whose only nodes are relevant to the goal list. This means that a much smaller game tree structure will be produced. Also, each goal will furnish a means for suggesting moves so that not all legal moves need be generated. The setting of a goal gives a fuzzy indication of the direction the game will take, which

provides for a more selective tree search than the evaluation function, and herein lies the advantage of the goal directed methods.

4.2 Previous Approach

Goal methods have been attempted before but did not come into wide use for game playing. Several plausible explanations are possible. People could have been preoccupied with the evaluation function to such an extent that other possibilities were ignored. Also, the initial poor showing of goal methods coupled with their apparent complexity might have discouraged workers in other fields. Finally, Newell et al. moved their goal approach to other areas thus leaving a void in game playing.

Newell, Shaw and Simon developed their program for playing chess in 1958. Since this is the only goal directed program written it will be described. At the beginning of each move a preliminary analysis routine produces an ordered (with respect to an unspecified measure of importance) static set of goals which are relevant to the starting position. It is this list of goals which now controls the remainder of processing. First of all, each goal has an associated base move generator which is responsible for suggesting initial moves toward the achievement of that particular goal. The tree structure growth is governed by a set of so-called analysis-move generators. The selectivity of these subroutines is based

on the generalization of a quiescent position. Only those moves are considered which radically affect the score for one or more goals in the predefined list. If no moves exist which do this, then a terminal position has been reached and the evaluation subroutine is used to generate a list of scores for each of the goals on the list which is backed-up by the alpha-beta procedure. This, then, is the essence of the operation of the program by Newell et al.

The program was not tested enough to determine the merit of goal techniques. Even so, it is possible to indicate several areas where weakness would exist. Because the program had only three goals to use, the selectivity of the preliminary analysis routine was not checked. Thus, given twenty or thirty goals it may not select the best ones each time, which in turn might produce poor quality moves. A more important weakness is that the goal list is fixed. This means that a faulty goal would not be changed, if detected, with the likely result of an adverse effect on the move generation procedure. Finally, the preliminary analysis routine was employed every time the program was to move. This implies the goal list could change completely at every move and means there was no continuity of planning from move to move. As expected, the program played the opening in an acceptable fashion since it only had goals associated with this phase of the game. Its performance over the remainder of the game was poor and after playing only a few games the program was abandoned and with it the

goal directed techniques as applied to games.

4.3 General Description of the System

In this section ideas will be presented which should remove the above named problems. These thoughts will be presented by describing how a program might be designed to solve the three problems of goal selection, implementation of a dynamic goal list and provision for continuity of planning.

Three main components will constitute the system, namely feature recognition programs, goal statements and planning programs. Feature recognition programs will check for the presence of various features in a given position by examining the "knowledge state" (defined below). From the list of features found in a position a number of goal statements will be constructed. The elaboration of these goals by planning routines will ultimately result in moves being proposed and the goal being made explicit. As moves are analysed, the feature recognizers will be used to evaluate the resultant position and, depending on their output, the goal list may be modified to reflect the intricacies of the position better. When the move has been determined the information about which goals are relevant and the plan or plans being attempted should not be discarded. This data will be used to prime the system on its next move thus giving it continuity of purpose.

Before describing the way in which each part of the

system should operate it must be made clear what is meant by the term "knowledge state". When given the initial position only the locations of the pieces constitute the initial knowledge state. However, as features are discovered and goals formed and elaborated these too become part of the knowledge state. Thus the knowledge state is the total amount of information known about the position, the system, and its operation up to its present state of execution.

4.3.1 Feature Recognition Programs

The purpose of these programs will be to query the knowledge state and output an evaluation of what was found. As an example, several feature recognition programs will be described. Many of them correspond to well-known chess principles.

1. Isolated Pawn - The presence or absence of isolated pawns will be indicated and if present then the locations of such will also be added to the knowledge state.
2. Material Balance - The current situation with respect to material will be given and all exchanges explored by means of a tree structure which indicates the effect of exchanges on material balance.
3. Useful Open File - Open files will be checked for by the corresponding feature recognition program and their presence recorded. The

utility of open files will then be assessed. Utility of a file may be determined by its nearness to the opponent's king, whether or not the enemy occupies it with a rook, which enemy pieces can be put en prise from squares along it or some combination of the preceding. If the utility is low, then the file will be recorded as not useful; otherwise, the file will be listed as useful and its important squares saved. A condition that the special squares be made safe will also be attached to those which are currently unsafe.

4. Achieve Goal <goal identity> - This program will check to see if the goal <goal identity> is currently being attempted and an indication given of whether or not it is being elaborated.

There will be many such feature programs, at least one for each chess principle. A fairly complete listing is to be obtained from the books My System [13] and Point Count Chess [8]. If those programs which check for various piece configurations employed a general structure for specifying them, it would then be possible for a single pattern matching routine to test for many different piece arrangements.

4.3.2 Goal Statements

These specify the goals which will be attempted. The way in which they will be created will be described in Section 4.4. For now, several examples should clarify their internal structure.

1. Goal - win black isolated pawn and (not white weak pawn structure or material loss for white)
2. Goal - protect white king and develop white pieces and (not black control of centre)
3. Goal - (white bishop vs black knight endgame or white outside passed pawn) and (not black centralized king)
4. Goal - white control of centre and (not black queen side majority unless black crippled queen side majority)

The goals to be attempted may in turn be thought of as a single goal - the goal to achieve as many goal statements as possible. Hence, this conception of goal statements enables one to visualize them as related and not independent.

4.3.3 Planning Programs

It will be the task of these programs to convert the goal statements of the type suggested in section 4.3.2 into action. Thus, these routines may be thought of as interpreting or elaborating the goal statements. The way in

which interpretation takes place will be described by following the elaboration of a sample goal statement. The statement used will be

Goal - white control of centre

- A. First of all, a move or several moves will be suggested which seem to aid in gaining control of the centre. Moves will be proposed for all goal statements to ensure the program has some move if all else fails.
- B. The goal white centre control will be subdivided into the subgoal (achieve white centre control from afar or achieve white centre control from near). A table listing the goal, its subgoals, and the relation required between them may be employed to effect this division.
- C. As the subgoals are obtained, the knowledge state will be checked to see if their achievement is likely. White centre control from afar is usually brought about by a bishop on the side of the board bearing on the centre. If both bishops are in the centre or have been captured, this means of centre control will be implausible and so the subgoal achieve white centre control from afar will be abandoned. A record of it will be maintained however, in case it becomes reactivated at some future time. In a similar fashion, achieve white centre control from near will be tested. If it is

likely to be achieved, it will be further subdivided and these subdivisions treated analogously. In general a goal will be divided into more specific subgoals. The subgoal with the highest apparent utility will be the one expanded next (this will be similar to MULTIPLE [17, Chapter 7]). This process will be continued until further elaboration of the goal white control of centre becomes less promising than further elaboration of some other major goal, or time runs out, or the goal has been proven attainable. One should note that as the goal statement is elaborated all subgoals will be saved. Abandoned goals will have the reasons for termination stored with them. As goals are elaborated, a means for evaluating moves will be obtained and the initial moves will be confirmed or replaced with more appropriate ones.

- D. When a goal statement appears to be unattainable, the next most likely one will be explored. Whenever a new subgoal (say G) is generated the relationship of G to previously terminated goals will be checked. Should such goals be related, they will be attempted along with the usual subgoals that would be tried by G. Thus, as each new piece arrangement is proposed, its use will be tested in other areas of failure.

- E. Also, each move generated will be evaluated by a standard list of feature programs to lessen the danger of a bad move being made which is the result of a deficiency in the original goal statement. This check would sometimes result in the addition of a condition to the goal statement and should help to eliminate blunders.

4.4 System Operation

The way in which these components interact will now be presented. The many feature recognition programs will be kept on an auxiliary storage device and the useful ones brought into main memory as needed. Inactive feature recognition programs will be tested while the opponent is moving and will replace useless routines. Thus, the feature programs will gradually change as does the position.

An overview of how the system works will now be described.

- A. The initial knowledge state of the system is the board description and the goal to make a move which improves the machine's position.
- B. The first goal causes information to be gathered about the position by execution of a list of feature recognition programs.
- C. Once information has been collected an initial goal statement list will be formed. This can be accomplished as follows:

- I. Those features which the program has and are good for it (as determined by a table look-up), will be kept as goals or traded for other good features.
 - II. Those features in the position which are bad for the program will be negated and an attempt made to attain this negation.
 - III. A similar procedure can be used to process the strong and weak points of the opponent except that here the goal will be to destroy the opponent's strengths and capitalize on his weaknesses.
- D. The planning programs will now elaborate the above goal statements until either a time limit expires or a move decidedly superior to other alternatives has been found. The quality of moves from the initial position will be assessed on the basis of the elaborated goals. The move that is made will be the one which seems to promote the most valuable goals.
- E. When a move has been selected, its causative goals and their elaboration should be saved. After the opponent's reply they will be used to give the system an idea of what has taken place in the past. Hence, it need not start its analysis afresh at every move.

4.5 Conclusions

The first attempt at a goal directed program for playing chess was abandoned before any general conclusions about the usefulness of the method could be made. Several possible areas of weakness were outlined in Section 4.2.

It is believed that these weaknesses will be overcome by the approach just outlined. By allowing the goal list to be modified it will be possible for the initial goal statement to be lacking a subgoal and have it added at a later time during execution. Also, by having the system save the goals it was trying to achieve when a move was made, continuity of purpose should be detectable. Another advantage of this approach should be continuous transition from one phase of the game to another. There will be no arbitrary division of the game into opening, middle and end games. This will occur naturally as feature recognition programs pass between passive and active memory. The program will not be designed to perform any kind of learning. Thus it will make the same move given the same position. This is a weakness of not only this program design but most other current programs. All in all, the system as described should be capable of playing better than current programs and with a substantially reduced game tree.

Chapter 5

PAST, PRESENT AND FUTURE

5.1 Past

Since this area has been discussed at length in Chapter 1 only the highlights will be mentioned. The introduction of the polynomial evaluation function and its extensions to learning by coefficient adjustment and signature tables were of major importance. Heuristic tree pruning techniques and backing up procedures were also noteworthy achievements. Finally, even though goal directed methods were not extensively applied to chess, they too have made an important contribution.

5.2 Present

At present not much work has been done with respect to the introduction of new ideas. For the most part, people have been interested in the extension of the polynomial evaluation function. There has also been some interest in learning; however, success in both these areas has been limited. The problem appears to be that, although many ideas already developed in other game playing areas seem useful, too few people have shown interest in testing their applicability to chess. For instance, the M & N backing up procedure was shown to be much more powerful than the minimax one; however, no one to date has tried to use it in

a chess playing program. Signature tables also fall into this category as do goal directed methods. It appears that once some people have written a chess playing system the authors are willing to make relatively minor changes but not major ones. It is also probable that some people are not aware of the above ideas when they program their own. This argument can be seen better in a paper by David Levy [10]. Basically, people are not following the steps suggested by I. J. Good [3] when devising a chess playing algorithm.

5.3 Future

There is much that can be done in the area of game playing and chess playing in particular. Theoretical and empirical studies should be carried out to determine the limits of current techniques and the conditions that must be satisfied for these methods to be successful. It is believed by this author that a polynomial scoring function approach to game playing will not produce a master calibre player in games such as Go and Chess. This also implies that the form of learning which is achieved by coefficient manipulation is insufficient to improve the calibre of a program. The goal approach should be pursued further and the prospects for developing an improved learning program explored. These will be active areas for future research.

5.4 Conclusions

Current techniques appear to be reaching their limit and so a new approach is needed. The evaluation function summarizes information too quickly and, in conjunction with forward pruning, there is a good chance of discarding a relevant move. For the development of more powerful programs, a method based on utilizing more of the information available from a chess position is necessary. The goal methods of Chapter 4 are a step in the right direction; however, a learning program would be the ultimate achievement. Certain forms of learning could be attempted with the goal methods of the preceding chapter, although suggestion of such was carefully avoided. The problem of learning is beyond the scope of this thesis and should receive separate study. It is time to stop developing new chess playing programs, for the moment, and to conduct studies to determine the usefulness of present methods and investigate the problems of developing a learning program based on goal methods.

References

- [1] Botvinnik, M.M.,
Computers, Chess, and Long Range Planning,
Springer-Verlag, 1970.
- [2] Gillogly, James,
"The Technology Chess Program", Carnegie-Mellon
University, AD-736043, November 1971.
- [3] Good, I.J.,
"A Five-Year Plan for Automatic Chess", Machine
Intelligence 2, American Elsevier, 1968, pages 89-
116.
- [4] Greenblatt, R.D., Eastlake, D.E., and Crocker, S.D.,
"The Greenblatt Chess Program", AFIPS Conference
Proceedings, volume 31, 1967, pages 801-810.
- [5] de Groot, A.D.,
Thought and Choice in Chess, Mouton, 1965.
- [6] Hanauer, Milton L.,
Chess Made Simple, Doubleday & Company, 1957.

- [7] Helms, H.,
The Book of the New York International Chess
Tournament 1924, Dover, 1961.
- [8] Horowitz, I.A., and Mott-Smith, G.,
Point Count Chess, Simon and Schuster, 1960.
- [9] Kozdrowicki, E.D., Cooper, D.W., and Licwinko, J.S.,
"Algorithms for a Minimal Chess Player : A Blitz
Player", International Journal of Man-Machine
Studies, volume 3, 1971, pages 141-165.
- [10] Levy, D.N.,
"Computer Chess - A Case Study on the CDC 6600",
Machine Intelligence 6, American Elsevier, 1971,
pages 151-162.
- [11] Newell, A., Shaw, J.C., and Simon, H.A.,
"Chess Playing Programs and the Problem of
Complexity", IBM Journal of Research and
Development, volume 2, October 1958, pages 320-
335.
- [12] Nilsson, Nils J.,
Learning Machines, McGraw Hill, 1965.

- [13] Nimzovich, Aron,
My System, David McKay Company, 1970.
- [14] Samuel, A.I.,
"Some Studies in Machine Learning Using the Game of Checkers", IBM Journal of Research and Development, volume 3, July 1959, pages 211-229.
- [15] Samuel, A.I.,
"Some Studies in Machine Learning Using the Game of Checkers, II - Recent Progress", IBM Journal of Research and Development, volume 11, November 1967, pages 601-617.
- [16] Shannon, Claude E.,
"Programming a Digital Computer for Playing Chess", The Philosophical Magazine, volume 41, March 1950, pages 356-375.
- [17] Slagle, James R.,
Artificial Intelligence : The Heuristic Programming Approach, McGraw Hill, 1971.
- [18] Slagle, James R., and Dixon, J.,
"Experiments With Some Programs That Search Game Trees", Journal of the ACM, volume 16, April 1969, pages 189-207.

- [19] Slate, D.J., Atkin, L.R., and Gorlen, K.E.,
"Chess 3.5", a user guide issued by Northwestern
University, 1971.

APPENDIX A The Alpha Beta Algorithm

A.1 Assumptions

1. Sprouting is depth first
2. Backing-up is the minimax method
3. Range(R) of scoring function satisfies the condition
 $-(M-1) \leq R \leq M-1$ for some real number M
4. The root node is a MAX node

A.2 Algorithm

1. With an associated beta value of M, set up a dummy MIN node whose only successor is the root node.
2. The initial alpha value for the root node is -M
3. When a MAX(MIN) node is sprouted its initial alpha(beta) value is that of the MAX(MIN) node 2 levels above it.
4. Before sprouting further branches from a MAX(MIN) node, compare the value of alpha(beta) at that node with the value of beta(alpha) at the MIN(MAX) above it. Proceed to sprout if $(\alpha < \beta)$. If $(\alpha \geq \beta)$ a beta(alpha) cut-off occurs and no more branches are sprouted from that MAX(MIN) node. The value of the MAX(MIN) node can be ignored when backing-up values to the nodes above.
5. As soon as a value for a MAX(MIN) position has been found this becomes the value of beta(alpha) at the MIN(MAX) node above it unless there is already a smaller(larger) beta(alpha) value at that node.

APPENDIX B Construction of Various Scoring Functions

First of all, 378 chess positions and the move a master made in each of them were obtained. Then, all legal moves from each position were listed and the scores of eleven features, used by WITA, obtained for each move. The three evaluation functions used to obtain Fig. 6 were constructed from the following information.

B.1 Features

1. The number of squares controlled by the opponent
2. The number of squares controlled by the computer
3. The number of squares the opponent can reach
4. The number of squares the computer can reach
5. A measure of material gain expected by the opponent
6. A measure of castling (-1 for unsafe, 0 for impossible, 1 for safe)
7. (Value of pieces attacked by the computer) - (Value of pieces attacked by the opponent)
8. (Value of pieces defended by the computer) - (Value of pieces defended by the opponent)
9. Difference in material value plus a penalty to prevent exchanging when the computer is losing material
10. Value of computer's largest piece that is en prise
11. Value of 2'nd largest opponent piece that is en prise

B.2 Notation

$F[k]$ is the value of the k 'th feature component

$X[i,j,k]$ is the value of the k 'th feature component
for the j 'th move in the i 'th position

$J[i]$ is the number of moves in the i 'th position

$X[i,k]$ is the value of the k 'th feature component for
the master's move in the i 'th position

B.3 Computation of Coefficients

SLAGLE METHOD [17, Chapter 11]

This computation is easy to perform and has been used with success by Samuel [15]. Essentially, one compares the components of the feature vector for the master's move with the corresponding components of the other feature vectors. The associated coefficient is increased if the master's component is better, unchanged if both components are equal and decreased otherwise.

1. $M \leftarrow 0$

2. $P[k], N[k], T[k] \leftarrow 0$ $k=1,2,\dots,11$


```

3.  T[k] <-- T[k] + |X[i,k] - X[i,j,k]|      i=1,2,...,378
    if X[i,k] > X[i,j,k] then P[k] <-- P[k] + 1
                                     j=1,2,...,J[i]
    if X[i,k] < X[i,j,k] then N[k] <-- N[k] + 1
                                     k=1,2,...,11

```

$$4. \quad M \leftarrow M + J[i] - 1 \quad i=1, 2, \dots, 378$$
$$5. \quad C[k] \leftarrow (M / T[k]) * (P[k] - N[k]) / (P[k] + N[k])$$

$$k=1, 2, \dots, 11$$

The desired coefficient vector is $C[k]$

QUADRIC METHOD [12, pages 27-30]

All pairs of features plus all single features are taken into account by this function. Slagle's method was used to adjust the coefficients. In Fig. 6 one should note that this function did not perform much better than the linear one. Since both signature tables and the above method take feature interaction into account it is doubtful if the signature table evaluation method will be of any use in chess.

```
1. F'[i * (23 + i) + j - 11] <-- F[i] * F[j]
                                     i=1,2,...,11
                                     j=i,i+1,...,11
```


2. $F'[i + 66] \leftarrow F[i]$ $i=1,2,\dots,11$

3. Apply the Slagle Method above, treating $F'[k]$ as the feature vector with 77 components. Thus the range of k is altered from 1-11 to 1-77.

NORMAL PATTERN METHOD [12, pages 55-58]

Assuming the feature components are samples from some normal distribution another form of scoring function may be applicable. The optimum classifier for normal patterns (feature vectors in this case) is both well known and widely used. For these reasons it seemed worthwhile to test this function.

1. $M \leftarrow 0$

2. $U[k], U'[k] \leftarrow 0$ $k=1,2,\dots,11$

3. $R[k,t], R'[k,t] \leftarrow 0$ $k=1,2,\dots,11$
 $t=1,2,\dots,11$

4. $U[k] \leftarrow U[k] + X[i,j,k]$ $i=1,2,\dots,378$
 $j=1,2,\dots,J[i]$
 $k=1,2,\dots,11$

5. $U'[k] \leftarrow U'[k] + X[i,k]$ $i=1,2,\dots,378$
 $k=1,2,\dots,11$
6. $M \leftarrow M + J[i] - 1$ $i=1,2,\dots,378$
7. $U[k] \leftarrow (U[k] - U'[k]) / M$
 $U'[k] \leftarrow U'[k] / 378$ $k=1,2,\dots,11$
8. $R[k,t] \leftarrow R[k,t] + (X[i,j,k] - U[k]) * (X[i,j,t] - U[t])$
 $i=1,2,\dots,378$
 $j=1,2,\dots,J[i]$
 $k=1,2,\dots,11$
 $t=1,2,\dots,11$
9. $R'[k,t] \leftarrow R'[k,t] + (X[i,k] - U'[k]) * (X[i,t] - U'[t])$
 $i=1,2,\dots,378$
 $k=1,2,\dots,11$
 $t=1,1,\dots,11$
10. $R[k,t] \leftarrow R[k,t] - (X[i,k] - U[k]) * (X[i,t] - U[t])$
 $i=1,2,\dots,378$
 $k=1,2,\dots,11$
 $t=1,2,\dots,11$

11. $R[k,t] \leftarrow R[k,t] / (M - 1)$ $k=1,2,\dots,11$
 $R'[k,t] \leftarrow R'[k,t] / 377$ $t=1,2,\dots,11$

12. $S \leftarrow (R)^{-1}$
 $S' \leftarrow (R')^{-1}$

13. The matrices S, S' are the end result of the computation. They can be used to compute the value(V) of a position as follows :

a) $V \leftarrow 0$

b) $V \leftarrow V + S[k,t] * (F[k] - U[k]) * (F[t] - U[t])$
 $\quad - S'[k,t] * (F[k] - U'[k]) * (F[t] - U'[t])$
 $k=1,2,\dots,11$
 $t=1,2,\dots,11$

B30033